CS3460: Data Structures Fall 2021
Assignment 04: Fun with Hash Tables
Due Date: Wednesday, October 13; 23:00 hrs
Total Points: 20

In this assignment, we will be building hash tables, and will use them to develop a simple spell checker application. We will also use spatial hashing to find the closest pair of points among a set of points in the two dimensional plane.

1. **Spellchecker (10 points):** We will be building a rudimentary spell checker – one that is capable of suggesting alternate spellings to words that are mis-spelled by exactly one character. This is achieved by first building a string set data structure implemented as a hash table, that is capable of storing all words in the English dictionary. It will then be possible to query this data structure for all possible alternate spellings of a word.

   Getting Started: Please download the files `StringNode.java`, `StringSet.java`, `SpellChecker.java` and `dictionary` into your working directory. The `StringNode` class provides the basic node structure for this assignment. It is already implemented with a basic constructor and appropriate get and set methods. You should be able to work this assignment without modifying this class. But feel free to modify it with additional methods if you think it will be beneficial for your implementation of the spell checker.

   The StringSet class: Complete the `StringSet` class, so that it fully supports operations `insert`, `find` and `print`. You also need to complete the method `hash` that hashes an input String to valid table indices using a polynomial hash function as discussed in class. You may evaluate this polynomial at any random prime number in the range $0 \ldots 10000$. For a list of prime numbers, see this Wikipedia page.

   Note that the `StringSet` class contains a constructor that only allocates a table of size 100. If the number of elements stored in this table reaches its size, then we will expand the table to twice its original size, and re-hash all the elements. The code for expanding the table size should be written in the `insert` function, at the appropriate check:

```
if (numelements == size) {
  // code for expanding the table.
  ...
}
```

   You may try to implement the above expansion code after you have tested all the other methods (including `insert`) in the `StringSet` class. The class should still be functional without the expansion code, but will be less efficient.

   SpellChecker Class: This class contains the `main` method of the spell checker application. A lot of code is already provided to get you started on the application. Specifically, an object of your `StringSet` class is declared, and is loaded with words from the dictionary. If the table expansion code is written correctly, this should take less than a second to execute.

   The main while loop waits for user input. Please finish up the code that provides alternate spellings to the user input that differ in at most one character. Note that only words that can be obtained by modifying at most one character are suggested, not words that are obtained

by adding or removing characters. Please also note that that the dictionary contains words with special characters and capital letters. You may ignore such words, and assume that the user enters a word in all lowercase letters. So you only need to suggest alternate spellings that contain only lowercase letters. The output should provide all alternate words, one per line. An example output is shown below.

```
Dicitonary loaded...
algorithm
algorithm is correct.
algorithx
Suggesting alternatives ...
algorithm
coee
Suggesting alternatives ...
code
coke
come
cone
cope
core
cote
cove
coed
```

Note: Pressing `Ctrl-d` marks the end of input and terminates the main while loop. You may also find the `StringBuffer` class in the Java library useful, particularly its `setCharAt` method.

2. **Closest Pair of Points (10 points):** In this problem, we will be computing the closest pair of points among one million points on the 2D plane using spatial hashing. Finding the closest pair of points has many applications; in machine learning classification, hierarchical clustering, collision detection in games and more.

   For this problem, you are not provided with any helper code, and you need to develop your code from scratch. The input is given in the file `points.txt` that contains one million points, described by its `x` and `y` coordinates on the unit square (each `x` and `y` coordinate lies in $[0, 1]$). The file contains each `x` and `y` coordinate of a point in a line, separated by a space. The distance between two points (x_1, y_1) and (x_2, y_2) is given by the following formula:

   $$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

   The naïve solution to this problem is to compute the distance between every pair of points, and always keeping track of the minimum distance computed. This takes $\Theta(n^2)$ time, and is infeasible in our case, where $n = 1000000$. Instead, we divide the unit square into $b \times b$ grids, each of size $\frac{1}{b} \times \frac{1}{b}$. Each point is therefore hashed into its specific grid (How?), and need to be only compared against points in its grid and the neighbouring 8 grids. This significantly reduces the number of pairs of points we look, provided we choose a good grid size. Feel free to experiment with different choices for the value of b.

Each of these grid cells should contain a linked list to store all the points hashed into it. If you used a class `Node` to store the points, then you may need to declare the two dimensional array of grid cells as:

```
Node [] [] grid;
```

and in your constructor, allocate memory for it.

```
grid = new Node[b][b];
```

Your program should roughly follow these steps:

(a) Allocate a 2D array of grid cells, for a particular choice of b.

(b) Read the input file, and hash each point to its corresponding grid cell.

(c) For each point, compare against all points in its grid cell and neighbouring grid cells, always maintaining the minimum distance computed.

(d) Print the minimum distance.

Please name your program `Closest.java`. The correct answer is given in the file `closest.out`. Please also use this to format your output accordingly.

Submission: Please submit all the files as a single zip archive `h04.zip` through ASULearn. The zip archive should only contain the individual files of the assignment, and these files should not be inside folders. Moreover, please do not include other extraneous files. Only include all files that belong to your solution.

Input/ Output Instructions: For all programs, until and otherwise stated, we will be taking the input from standard input (`System.in`) and will be sending the output to standard output (`System.out`). Since we will be using an automatic grading system, please make sure that your output format exactly matches the description above. So make sure that there are no extraneous output in terms of spaces, newlines and other characters.

Notes on Coding: Please do not include user-defined packages in your code. Your code should run in the Unix/Linux machine using the commands `javac` and `java`.

Please note that you are **not allowed to use Java Collections** which contain pre-defined libraries for many of the data structures that we learn in class. The purpose of these assignments is to build these data structures from first principles. Programs that includes these objects will receive 0 points.